Tutorial for 2D Scene Graph and Nodes

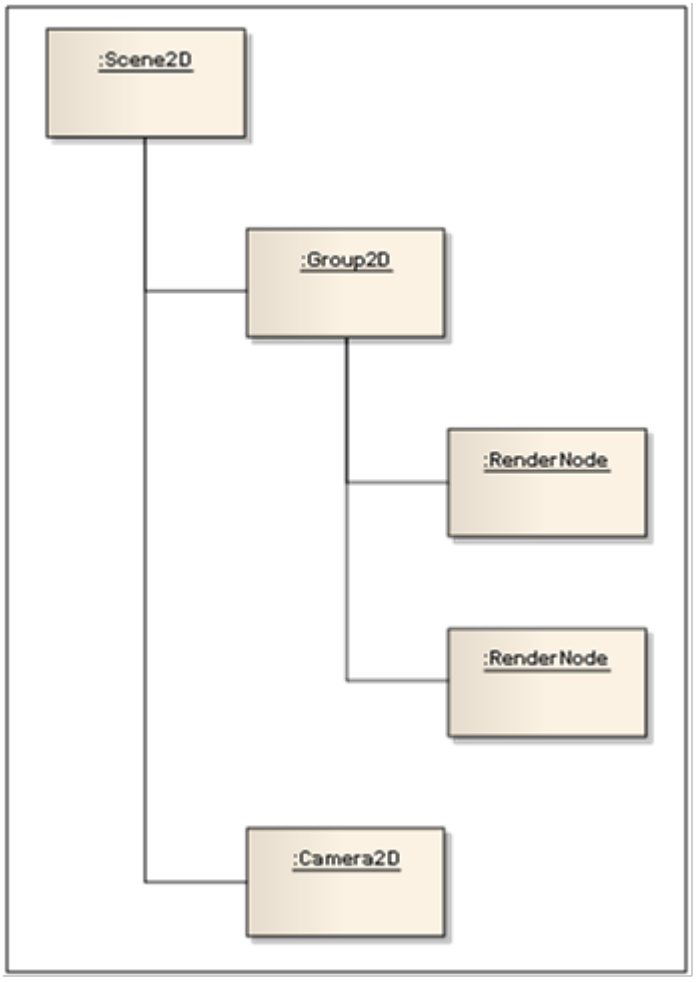
- 2D Scenes and Nodes
- 2D Node Transformations
- 2D Effects
- 2D Scene Graph Dynamics
- Camera / Viewport
- 2D Layout
- Candera 2D Listener

2D Scenes and Nodes

Scene Graph

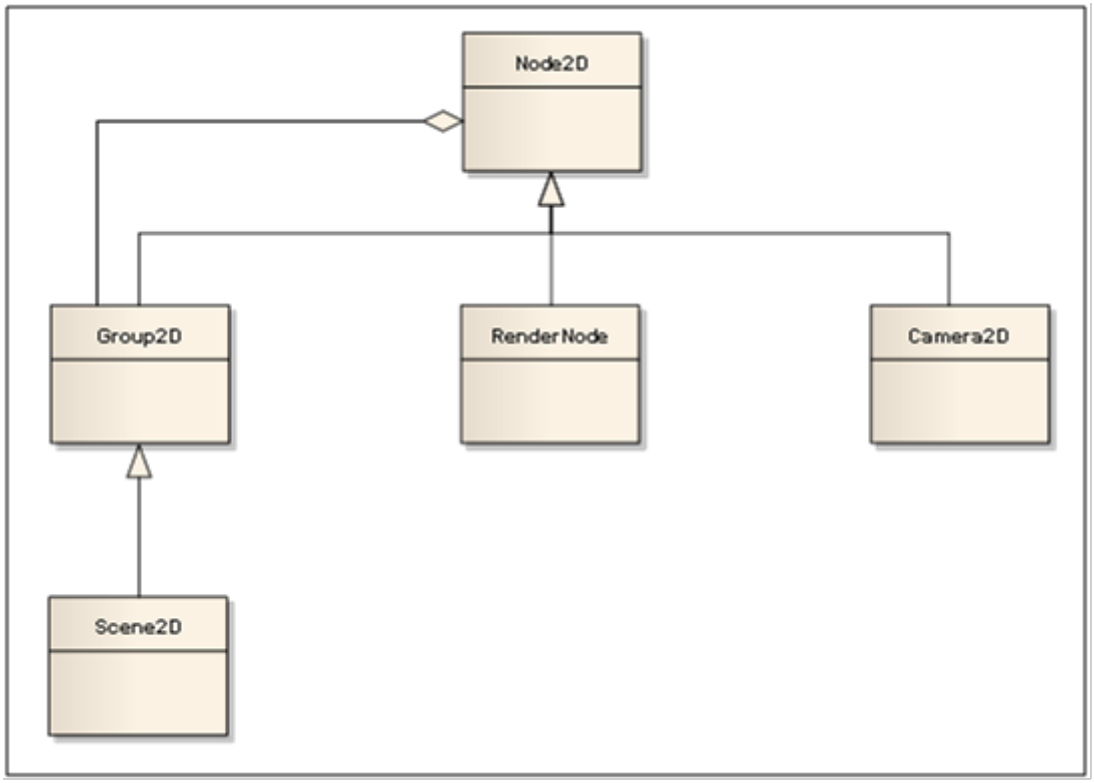
The 2D scene graph is a tree structure which organizes nodes of type Candera::Node2D. The root of each scene graph has to be a node of type Candera::Scene2D. All nodes of type Candera::Node2D can hold sub nodes added by Candera::Node2D::AddChild().

Nodes of type Candera::RenderNode are representing content which has to be rendered to the target, e.g. text or a bitmap. At least one node of type Candera::Camera2D which is the link to the render target (e.g. a display) has to be part of the scene graph.



RenderNode

Only nodes of type RenderNode are considered during rendering. Each render node requires a list of effects (effect chain) which implements the visual representation of that node. Each render node has a depth value as parameter which defines the order of rendering (z-order). Higher value means background, lower value means foreground.



Transformations

The functionality to define the local coordinate system is derived from Candera::Transformable2D and consists of following components:

- position,
- rotation,
- scale,
- and the pivot offset.

This information is always relative to its parent node. Changing one of these parameters affects the presentation of all child nodes. If for example a node is rotated, all child nodes are rotated accordingly. Each nodes rotation, translation (position) and scale values are combined in one transformation matrix which is multiplied with the parent nodes transformation matrix.

2D Node Transformations

2D Node Manipulation Example

To illustrate basic node transformation operations, the *NodeManipulationWidget_2D* and the *DisplayPositionWidget2D* part of the Tutorial Widgets cand be used. An usage example for *NodeManipulationWidget_2D* can be seen in **NodeManipulationSolution_2D** solution provided in the content folder of *cgi_studio_player*.

Please consider:

- A *Node2DToTransform* on which the following transformations are made has to be selected in the widget.
- Only if the widget is *enabled* (this property is also derived from a base class), the setter methods of the widget properties will take effect.
- If a widget property setter changes a node property of the associated node, the visual effect can be seen, but note that the static node properties maintained by SceneComposer are not changed accordingly!
- Widget dynamics are never reflected to static scene structure. It is recommended to use SceneComposer only for property configuration and dynamics only in the Player.

Translations

Once the NodeManipulationWidget_2D is linked with a node from the static scene tree and enabled, the position coordinates can be set in the Player for this property to change the position.

```
            // Set node position
            m_node->SetPosition(m_position);    // set absolute node position
            // End Set node position
```

```
            // Translate node
            m_node->Translate(m_translate);     // translate actual node position
            // End Translate node
```

> Candera::Transformable2D::SetPosition sets the absolute position of a node compared to Candera::Transformable2D::Translate, which adds the given vector to the current position. So setting the position first and then translating the node gives a different

> behavior than vice versa.

Rotations

Similar to translation, rotation can also be applied via Candera::Transformable2D::Rotate or Candera::Transformable2D::SetRotation.

```
            // Rotate node
            m_node->Rotate(m_rotate);
            // End Rotate node
```

Scaling

Again there is also a Candera::Transformable2D::Scale and Candera::Transformable::SetScale method.

```
            // Scale node
            m_node->Scale(m_scale);
            // End Scale node
```

Pivot Point

It is possible to change your pivot point with the Candera::Transformable2D::TranslatePivotPoint (or Candera::Transformable2D::SetPivotPoint) method. The pivot point can be used to define e.g. the rotation point that the node should rotate around. It affects the scaling too.

How to get the screen space coordinates of a node

With the DisplayPositionWidget2D you can retrieve the screen coordinates of an associated node. Please consider that a node and a camera have to be selected in the widget. In the solution the widget is disabled by default, so if you want to see an output of your display position, you need to enable it. To get the screen space coordinates of an associated node, you simply need to add the render target position of your node to the window position.

- First you need the world position of your node. With this position you can retrieve the viewport position, which you need for retrieving the render target position.

  ```
            Vector2 worldPosition = node->GetWorldPosition();
            Vector2 viewportPos = Math2D::TransformSceneToViewport(*camera, worldPosition);
            Vector2 renderTargetPos = Math2D::TransformViewportToRenderTarget(*camera, viewport
  ```

- Now you need the window.

```
        Window* w = camera->GetRenderTarget()->GetGraphicDeviceUnit()->ToWindow();
        if (w != 0) {
            Vector2 delta(static_cast<Float>(w->GetX()), static_cast<Float>(w->GetY()));
```

- With these two Candera::Vector2 you can calculate the screen space Candera::Vector2 of your node.

```
            Vector2 displayPos = delta + renderTargetPos;
```

- displayPos now contains the X and Y screen space coordinates of the selected node.

2D Effects

Description

Candera2D renders content exclusively by pieces of code which are named effects. Effects represent semantic functionality which considers hardware and driver capabilities.

> The set of effects supported depends on the feature set of the underlying graphic device unit. This tutorial is based on the reference platform ***iMX6***. Refer to the CanderaPlatformDevice API documentation to learn about the set of 2D effects supported on the platform used.

Example Solution

- 2DEffects Sample Solution in SceneComposer.

2D Effect Types

Effects can be distinguished in three types of operations:

1. Candera::BrushEffect2D provides means of generating graphical content out of user-defined data. An example would be an effect which generates a rectangle of solid color based on red, green and blue values. Brush effects generate pixel data as output.
2. Candera::InPlaceEffect2D requires input data provided. Input data is always given in form of pixel data and is supposed to be provided by either a Brush Effect or another InPlace Effect. These kinds of effects can utilize any additional parameters to transform the pixel data to output data. The output is again pixel data.
3. Candera::BlendEffect2D is very similar to InPlaceEffect2D. To be specific, every InPlace effect resembles a specialization of blend effect, namely that they blend with a fixed functionality. What makes Blend Effects special is only their purpose of being the effect which also takes the render target into account as an input. This allows *blend effects* to transform the input and the render target to any kind of combination of both. Output is again pixel data.

The effect types can be linked together so that they resemble an **Effect Chain**. An effect chain consists of one brush effect, zero to n InPlace effects and one blend effect.

- The Brush Effect references already existing pixel data, so it defines the source data of the node (bitmap, solid color, text).
- InPlace Effects define pixel manipulation functions like mask, color transformation, etc.
- The Blend Effect is attached at the end of the chain so that the pixel data generated by the Brush Effect is processed together with the render target's pixel data to a combination.

Combined Effects

A Combined Effect is a single effect which represents a whole sequence of other effects in an effect chain. For example, if you want to render bitmap data with an alpha function, there usually is a combined effect which represents the Brush Effect and the Blend Effect in one single object.

- Combined Effect for Brush and Blend Effect: Candera::BitmapBrushBlend.

This single object can take the advantage to perform operation in a single step as the external interfaces (pixel data) can be replaced internally by setting options in the native rendering API.

Refer to Candera::CombinedEffect2D about which CombinedEffects are supported by Candera.

Combined Effects Examples

An example of various effects and how they are configured can be seen in *2DEffects* Sample Solution in SceneComposer.

Bitmap Effects

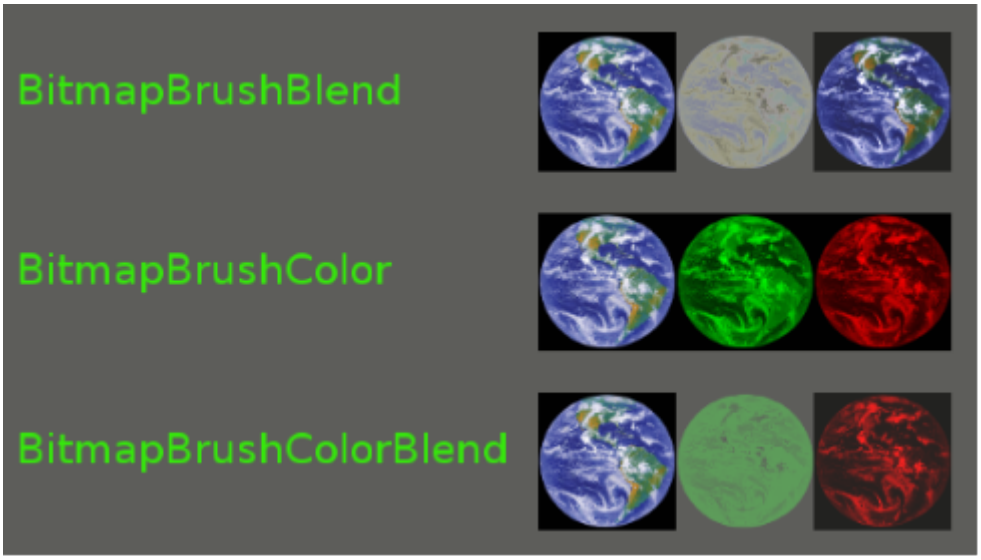
In order to render an image, a **BitmapNode** with one of the following effects must be set in a 2D scene:

- Candera::BitmapBrush
- Candera::BitmapBrushBlend
- Candera::BitmapBrushColor
- Candera::BitmapBrushColorBlend

For the above effects, the following properties can be set to obtain different rendering results:

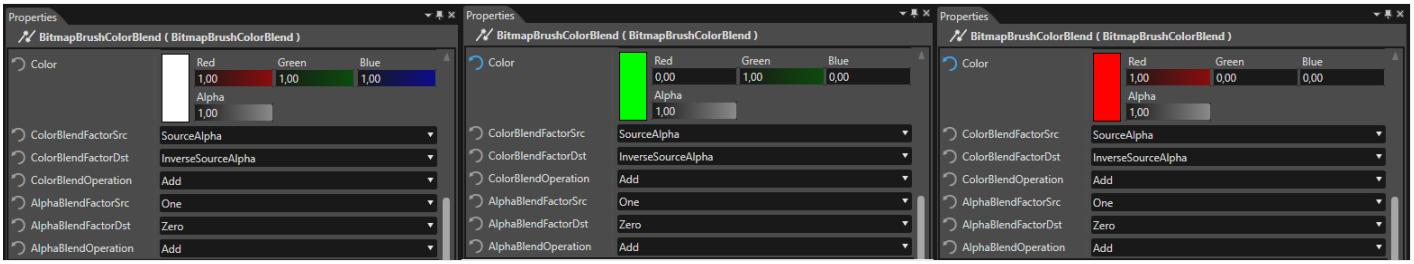
- **Color** - values for red, green, blue and alpha must be set
- **Color Blend Factor** for source and destination
- **Color Blend Operation**
- **Alpha Blend Factor** for source and destination
- **Alpha Blend Operation**

An example of Bitmap effects with different properties is shown below:



Candera::BitmapBrushColorBlend is a combined effect between Candera::BitmapBrushBlend and Candera::BitmapBrushColor. The properties of this effect sum up the properties of the other two.

The values for the properties for Candera::BitmapBrushColorBlend used in the example, for left most, centre and right most images, are:



Text with Bitmap Effects

In order to render text, a Text Node with one the following effect must be set in a 2D Scene.

- Candera::BitmapBrushBlend
- Candera::BitmapBrushColor
- Candera::BitmapBrushColorBlend
- Candera ::GlLinearGradientBitmapBrushBlend

Text node is a Render node that render text using the attached Bitmap Brush effect.

- Text and Style properties can be set using the Text node properties, as they are inherited from the Render node. To apply Style properties, the font of the text must be set.
- For Candera::BitmapBrushColorBlend, the properties Text Color, Color Blend Factors and Operations, Alpha Blend Factors and Operations, and the Text Clipping Area can be set to achieve different rendering effects.
- An example of Candera::BitmapBrushColorBlend and related properties is shown below.

Brush Effects

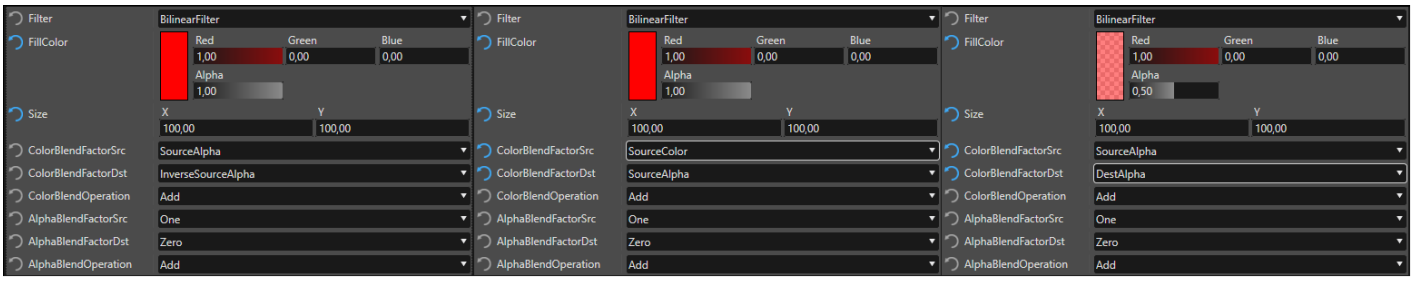
The following effects can be used with a **RenderNode** in a 2D scene:

- Candera::SolidColorBrush
- Candera::SolidColorBrushBlend

Fill Color must be set for the above effects.

For Candera::SolidColorBrushBlend, the properties **Color Blend Factor** and **Operation** and **Alpha Blend Factor** and **Operation** can also be set to obtain different rendering results.

An example of Candera::SolidColorBrushBlend and related properties is shown below:



Mask Effects

As the name suggests, Mask effects allows masking different areas of a given image by applying a mask image with transparent areas.

The following effects can be used with a **BitmapNode**:

- Candera::GlBitmapBrushColorMaskBlend
- Candera::GlBitmapBrushMaskBlend

For the above effects, the following properties must be set:

- **Image** – the image on which the mask will be applied
- **Mask** – the mask image with transparent areas
- **Mask Node** – node used to control the position of the mask. If set to 0, the position of the current node is used. For **iMX6**, the mask takes into account the scale, the rotation and the pivot of the node.

An example of mask effects is presented below:

Left most image has no **Mask Node** set, therefore the **BitmapNode** associated with the effect is used. The image from the center has **Mask Node** set to a **RenderNode** with **Rotation** property = -50. Thus, the mask also appears rotated. Right most image has **Mask Node** set to a **RenderNode** with different values for **Scale** property (x= 0,5 ; y = 0,5). Thus, the mask also appears smaller.

The properties **Color Blend Factor** and **Operation** and **Alpha Blend Factor** and **Operation** can also be set to obtain different rendering results.

Manipulating an Effect on a Render Node

To illustrate effect manipulation operations, the *NodeManipulationWidget_2D* part of the Tutorial Widgets in combination with the NodeManipulationSolution_2D provided in the content folder of *cgi_studio_player* can be used.

The type Candera::RenderNode defines a node that shall be rendered. Each render node requires an effect chain, which implements the visual representation of the node.

The following properties manipulate a Candera::SolidColorBrushBlend effect on a render node (this is a already combined effect with brush, inplace and blend):

- *RenderNodeToManipulate:* Select a RenderNode to manipulate its effect. Because of casts in the widgets code only select a render node with Candera::SolidColorBrushBlend, otherwise the casts will fail! (See Manipulating an effect)
- Set the widgets *SolidColor*, *FillArea*, *Blend* values to change the properties of the effect dynamically.

Manipulating an Effect

First the selected render node (which comes with type Candera::Node2D by the CDAProperty) is casted to a Candera::RenderNode.

```
    // Cast Node2D to RenderNode
    m_renderNode = Dynamic_Cast<RenderNode*>(node);
    // End Cast Node2D to RenderNode
```

Next the first effect of the selected render node of type Candera::Effect2D is casted to Candera::SolidColorBrushBlend to get access to specialised properties.

> This simple example is based on a RenderNode with SolidColorBrushBlend effect, so if the example solution is changed, please take care to use only SolidColorBrushBlend on the node to manipulate.

```
    // Get and cast first effect of render node
    Effect2D* effect2D = node->GetEffect(0);
    SolidColorBrushBlend* effect = Dynamic_Cast<SolidColorBrushBlend*> (effect2D);
    // End Get and cast first effect of render node
```

Now properties of the effect can be set.

```
        // Set solid color
        SolidColorBrush& brush = effect->GetSolidColorBrush();
        brush.Color() = m_color;
        // End Set solid color
```

```
        // Set fill area
        brush.Size() = m_rect;
```

```
        // End Set fill area
```

The effect should be updated each time a property is changed. A good place for updating the effect might be the widget Update method.

```
        // Update effect
        static_cast<void>(effect->Update());
        // End Update effect
```

Arbitrary properties on arbitrary effects can be manipulated the same way.

How to Integrate a Custom Combined Effect into SceneComposer

Integrate a custom effect to Candera and SceneComposer

- Create and implement your own effect.
- Integrate the effect into the effect library of the device.

For **DEVICE** that would mean to modify cgi_studio_candera/src/CanderaPlatform/Device/DEVICE/2D/DEVICEEffectLibrary.cpp, so that the new effect implementation is included and the Effect2DLibrary MetaInfo is extended by the new effect. (**DEVICE** stands for your specific device)

- Integrate the new device library in SceneComposer. Therefore set CMake source to *<cgi-studio-root>/cmake/Candera/Player* and build a SCHost.dll. The library CanderaDevice<device>.lib is linked to the SCHost.dll, which than has the new effect included.
- Start SceneComposer with the new SCHost.dll. The new effect is listed in the toolbox for a 2D scene.

How to create a combination of two effects

There are several effects which can be combined to a new effect. How to combine two effects to a new one is shown in Candera::GlBitmapBrushColorMaskBlend implementation.

For the new effect Candera::GlBitmapBrushColorMaskBlend you need a combination of Candera::BitmapBrushColorBlend and Candera::GlBitmapBrushMaskBlend. Create and implement the new effect analogous to Candera::GlBitmapBrushMaskBlend. Now apply following changes:

- The new effect contains both, MaskEffect and ColorEffect as member (wrapper, see in .h file).
- The implementation of the method Render() is a combination of Candera::BitmapBrushColorBlend::Render() and

Candera::GlBitmapBrushMaskBlend::Render(). This method has to handle all four effects in the following order:
    - blend effect,
    - color effect,
    - mask effect and
    - brush effect.
- Upload(), Unload(), and Update() have to handle all four effects.
- Apply changes to the Effect Meta Info. The new effect needs all four effects (BitmapBrush, ColorEffect, MaskEffect, BlendEffect).

2D Scene Graph Dynamics

Description

Usually all required nodes are already specified and preconfigured via SceneComposer, so in most cases it won't be necessary to make any adaptations within the scene graph loaded from an asset at runtime.

However, some use cases might require such adaptations. This tutorial covers the most common means to manipulate the scene graph structure dynamically.

Example Solution

- SceneGraphDynamicsSolution_2D in folder *cgi_studio_player/content/Tutorials/03_SceneGraphAndNodes*

Adding and Removing 2D Nodes

Scene Graph Dynamics 2D Example

There are some differences between a 2D and a 3D scene graph. This chapter will explain the differences in detail.

For 2D scene graph dynamics, the following examples can be used:

- **SceneGraphDynamicsSolution_2D** from folder *cgi_studio_player/content/Tutorials/03_SceneGraphAndNodes*
- **SceneGraphDynamicsWidget_2D**

Adding and Removing 2D Nodes

In the widget a RenderNode with text is created and added/removed to/from the 2D scene graph. For adding and removing 2D nodes on the scene graph the following properties can be used:

- **AddToNode2D:** Select the node where the new RenderNode should be attached. This node can be any sub-type of Candera::Node2D.

```
        CdaProperty(AddToNode, Candera::Node2D*, GetAddToNode, SetAddToNode)
            CdaDescription("Node2D, where a new TextNode shall be added to")
            CdaCategory("Adding and Removing 2D Nodes")
```

```
        CdaPropertyEnd()
```

- **TextNodeStyle :** Sets the style to be used for the text node. This allows the user to provide a custom style for the dynamic text.

```
CdaProperty(TextNodeStyle, Candera::TextRendering::SharedStyle::SharedPointer,
GetTextNodeStyle, SetTextNodeStyle)
 CdaDescription("Style for the new Text Node ")
 CdaCategory("Adding and Removing 2D Nodes ")
```

- **AddTextNode:** If AddTextNode is enabled, a new text node with the selected font is created and attached to the selected node. If disabled, the text node is removed from the scene graph again.

```
        CdaProperty(AddTextNode, bool, GetAddTextNode, SetAddTextNode)
            CdaDescription("If AddTextNode is enabled, a new TextNode with Selected
TextNodeStyle Will be added to the selected AddToNode property. If disabled the new TextNode
will be removed.")
            CdaCategory("Adding and Removing 2D Nodes")
        CdaPropertyEnd()
```

Creating a new Text Node

To display new text, a new TextNode2D is created. TextNode2D represents a Render Node (Candera::TextNode2D). It requires a BitmapBrush effect for rendering. The Candera::BitmapTextNodeRenderer then generates the bitmap images from the given text, and these images are finally rendered using the attached Bitmap Brush effect.

Create an instance of BitmapBrushEffect and a text node for the scene graph using the following code used.

```
BitmapBrushColorBlend::SharedPointer bbc = BitmapBrushColorBlend::Create();
bbc->GetColorEffect().Color().Set(Color(255 / 255, 255, 255 / 255, 1));
m_addedNode = TextNode2D::Create();
m_addedNode ->AddEffect(bbc.GetPointerToSharedInstance());
m_addedNode ->SetName("AddedTextNode");
m_addedNode ->SetText("AddedNode");
m_addedNode ->SetStyle(m_textNodeStyle);
m_addedNode ->SetTextNodeRenderer(BitmapTextNodeRenderer::Create());
```

```
m_addedNode ->SetPosition(100, 200);
```

Adding Text Node to Scene Graph

Next the text node is uploaded to VRAM and appended to the selected node.

```
        static_cast<void>(m_addedNode->Upload());
        static_cast<void>(m_addToNode->AddChild(m_addedNode));
```

Removing and Destructing Text Node

A 2D node can be removed by calling Candera::Node2D::RemoveChild. This is also described in the 3D part of this chapter (see Removing and destructing the Billboard). For removing and completely destructing the node, following code is used:

```
        static_cast<void>(m_addedNode->Unload());
        m_addedNode->Dispose();
        m_addedNode = 0;
```

The text node is unloaded from VRAM. The method Candera::Node2D::Dispose removes the node from its parent and frees the resources.

> In SceneComposer the manipulated scene graph and the changed properties do not appear because it is only changed dynamically by the widget.

> The widget must control memory and VRAM management of its internal dynamic subtree autonomously.

Replacing a 2D Effect

Replacing an Effect

In the SceneGraphDynamicsWidget_2D example, the effect of a Candera::RenderNode can be either replaced by a Candera::SolidColorBrushBlend or a Candera::BitmapBrushBlend effect. Of course the same procedure works for Text effects.

To replace the effect in the example, the following properties can be used:

- **NodeToChange:** Select any render node with an effect whose appearance should be changed. Ensure that just a render node which has an effect is selected (in the sample solution this could be e.g. the node BitmapNode_BarTop). For further information see

Adding Replacing Effect.

```
        CdaProperty(NodeToChange, Candera::Node2D* , GetNodeToChange, SetNodeToChange)
            CdaDescription("Select a RenderNode which effect should be exchanged")
            CdaCategory("Replacing a 2D Effect")
        CdaPropertyEnd()
```

- **ChangeEffect:** If ChangeEffect is enabled, the effect of the selected NodeToChange will be removed and a Candera::SolidColorBrushBlend effect is added to the node, instead. If disabled, the actual effect will be removed too and a Candera::BitmapBrushBlend effect is added to the node.

```
        CdaProperty(ChangeEffect, bool, GetChangeEffect, SetChangeEffect)
            CdaDescription("Change
effect of selected NodeToChange. Set either a SolidColorBrushAlphaBlend (enabled) or BitmapBrus
            CdaCategory("Replacing a 2D Effect")
        CdaPropertyEnd()
```

Creating a Color Effect

First a new effect is created by creating an instance of Candera::SolidColorBrushBlend.

> 2D effects are part of the device package, therefore in order to create a concrete instance of the effect, the effect must be enabled for the device. Please see chapter **2D Effects** of this tutorial to see how to enable an effect for a certain device.

```
        Candera::SolidColorBrushBlend::SharedPointer brush =
Candera::SolidColorBrushBlend::Create();
        brush->GetSolidColorBrush().Size().Set(m_Rect);
        brush->GetSolidColorBrush().Color().Set(m_ColorBlue);
        brush->GetBlendEffect().SetBlendMode(RenderDevice2D::SourceAlpha, RenderDevice2D::Invers
```

Adding Replacing Effect

Once the effect is created it can be added to the render node. In the following code snippet the selected NodeToChange is cast to a Candera::RenderNode (for this a RenderNode **must** be selected, otherwise the cast will fail). The first effect of the node is removed via its reference and the new created effect added.

```
    RenderNode* rn = Dynamic_Cast<RenderNode*>(m_nodeToChange);
    if (rn == 0) {
        return;
    }

    static_cast<void>(rn->Unload());

    //remove old effect
    Effect2D* oldEffect = rn->GetEffect(0);
    static_cast<void>(rn->RemoveEffect(oldEffect));

    //add new effect
    static_cast<void>(rn->AddEffect(effect));
    static_cast<void>(rn->Upload());
```

Creating a Bitmap Effect

Creating a Candera::BitmapBrushBlend effect is similar to creating other effects (see Creating a Color Effect). Just apply other properties.

```
        Candera::MemoryManagement::SharedPointer<Candera::BitmapBrushBlend> brush =
Candera::BitmapBrushBlend::Create();
        Bitmap::SharedPointer bitmap = Base::GetAssetProvider()->GetBitmapById(CgiAssetNames::cc
        if (m_image != 0) {
            static_cast<void>(m_image->SetBitmap(bitmap));
        }
        brush.GetSharedInstance().GetBitmapBrush().Image().Set(m_image);
        brush.GetSharedInstance().GetBlendEffect
().SetBlendMode(RenderDevice2D::SourceAlpha, RenderDevice2D::InverseSourceAlpha, RenderDevice2D:
```

> In SceneComposer the manipulated scene graph and the changed properties do not appear because it is only changed dynamically by the widget.

> The widget must control memory and VRAM management of its internal dynamic subtree autonomously.

> Don't forget to set a name for the dynamically created nodes.

Cloning 2D Nodes

Cloning 2D nodes is equivalent to cloning of 3D nodes. In accordance to Tree Cloning Strategies, the following support is provided for deep cloning:

- Candera::TreeCloner2D is a specialization for the 2D scene tree of Candera::TreeClonerBase.
- Candera::DeepNode2DCloneStrategy provides a generic node strategy that clones the delegates the cloning of derived Nodes to specialized strategies.
- Candera::DeepCompositeGroup2DCloneStrategy resolves node binding by use of Candera::TreeMatch2D.
- Candera::DeepRenderNodeCloneStrategy resolves the sharing of effects by use of a map.
- Candera::DeepTreeCloner2D is a wrapper of Candera::DeepNode2DCloneStrategy that provides an interface similar to Candera::TreeCloner2D.

Refer to

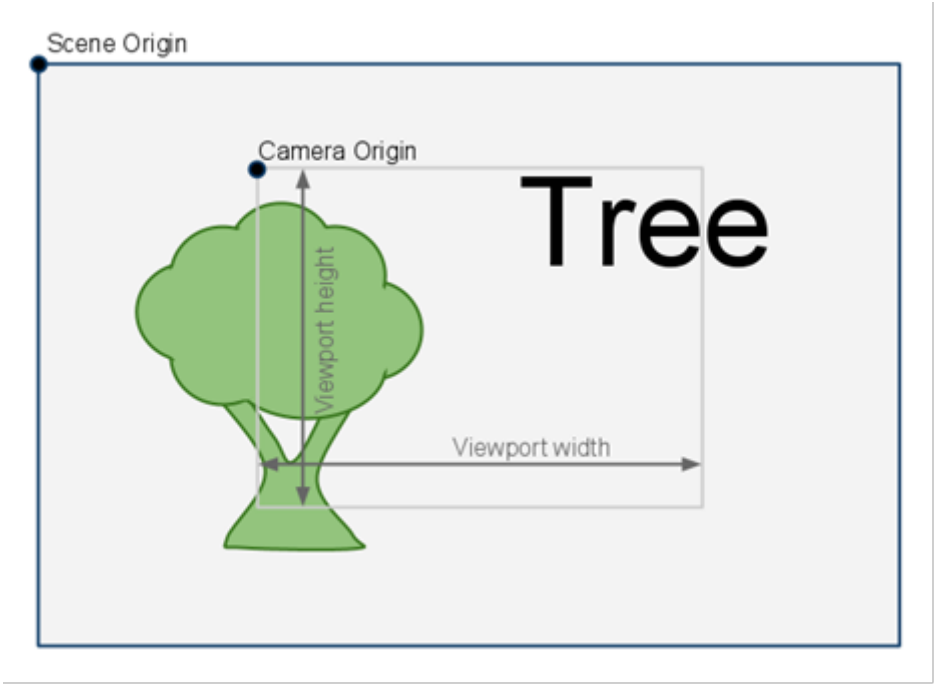
- Candera::Node2D::Clone

as well as the example for cloning 3D nodes:

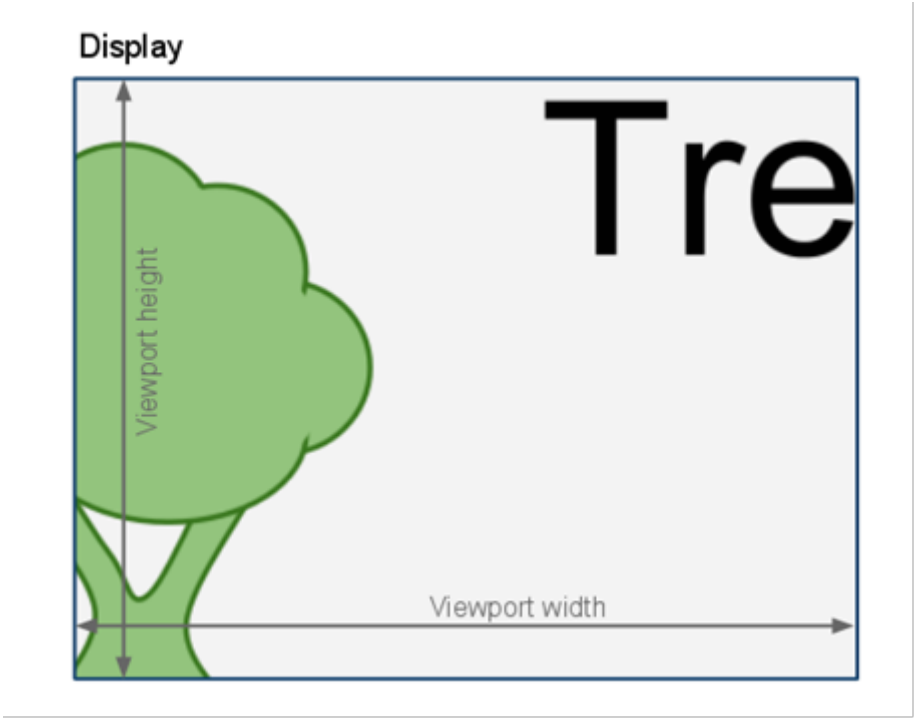
- Cloning 3D Nodes

Camera / Viewport

The following example shows the interaction between scene and camera. The objects are placed relatively to the scene origin (which is the root node). The camera defines the view to that scene by defining also a position relative to the scene.



The view port dimensions (width and height) relate to the area on the render target the camera is painting on.



2D Layout

Description

In a typical user interface users need to distribute screen space to different elements. If the size of elements is dynamic, it's desired that the screen space distribution is done automatically based on the contents's natural size. To enable easy arrangement of elements, Candera2D supports layout functionalities.

Example Solution

- Layout2DSolution in folder *cgi_studio_player/content/Tutorials/Layout2DSolution.zip*

Candera::GridLayouter

Introduction

Basically, in SceneComposer for each Candera::Group2D a Candera::Layouter can be attached. The attached layouter defines the arrangements of the group's children.

Cell Spanning

Candera::GridLayouter provides a feature that allows joining of adjacent columns and rows into a larger single cell.

Cells can be joined horizontally, vertically or both. In order to achieve this, you must specify **Row Span** and **Column Span** properties for elements inside a Candera::GridLayouter:

- Row Span: Specifies the number of rows that the element should span across.
- Column Span: Specifies the number of columns that the element should span across.

> Default value for row and column span properties is 1.

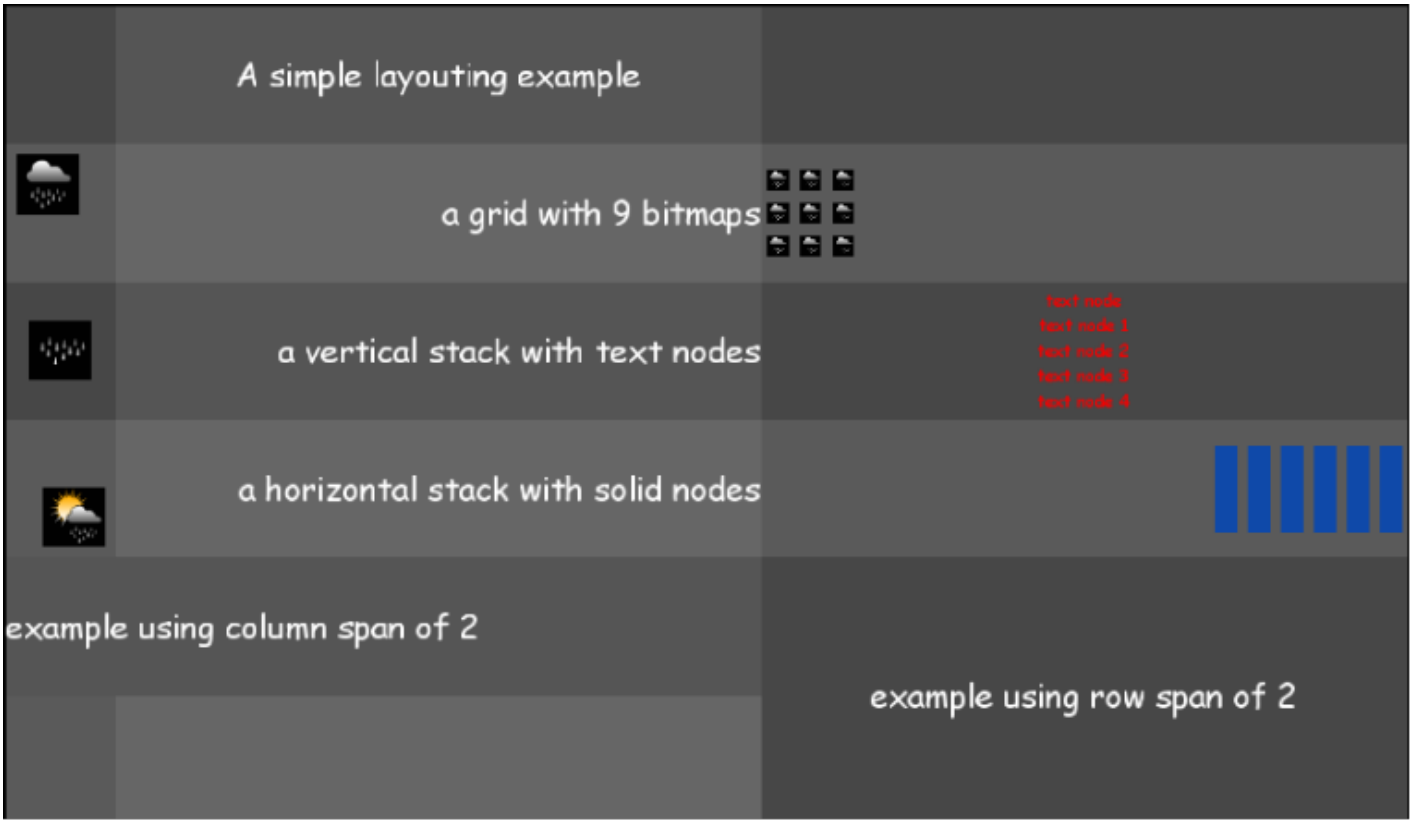
Example

The **Layout2DSolution** provided in the content folder of *cgi_studio_player* gives an example how the 2D layout can be applied in SceneComposer. This section gives just a brief overview on layout features and shows how it looks like.

In the example on the outermost group, a Candera::GridLayouter with 6 rows and 3 columns is defined.

All child nodes, here a bunch of bitmap-, text- and solid-nodes, are arranged along the grid. Each inner grid element can set various dynamic properties which define its size and alignment inside a grid cell.

Layouters can be nested. The outer grid contains a inner grid and two inner stack layouters (Candera::StackLayouter).



The example consists of a grid layouter with fixed size 1280 x 756 px.

- The **3 column's width** values are: 100 / 0,5 / 0,5.
  - The first column has a absolute width of 100 px,
  - the second and third column share the rest of the space (1180 px) relatively to each other (indicated by 0.0 <= value <= 1.0).
  - In fact the effective width column 2 and 3 is calculated by the layouter as follows: (1180 * 0,5) / (0,5 + 0,5) = 590 px.
- The **row heights** are defined as 126 / 0,1 / 0,1 / 0,1 / 0,1 / 0,1..
  - The first row has a absolute height of 126 px,
  - and the remaining height (630 px) is partitioned relatively to the other rows. Row height = (630 * 0,1) / (0,1 + 0,1 + 0,1 + 0,1 + 0,1) = 126 px.

> If a size value is set to -1 the width or height is set to the preferred size of the inner elements.

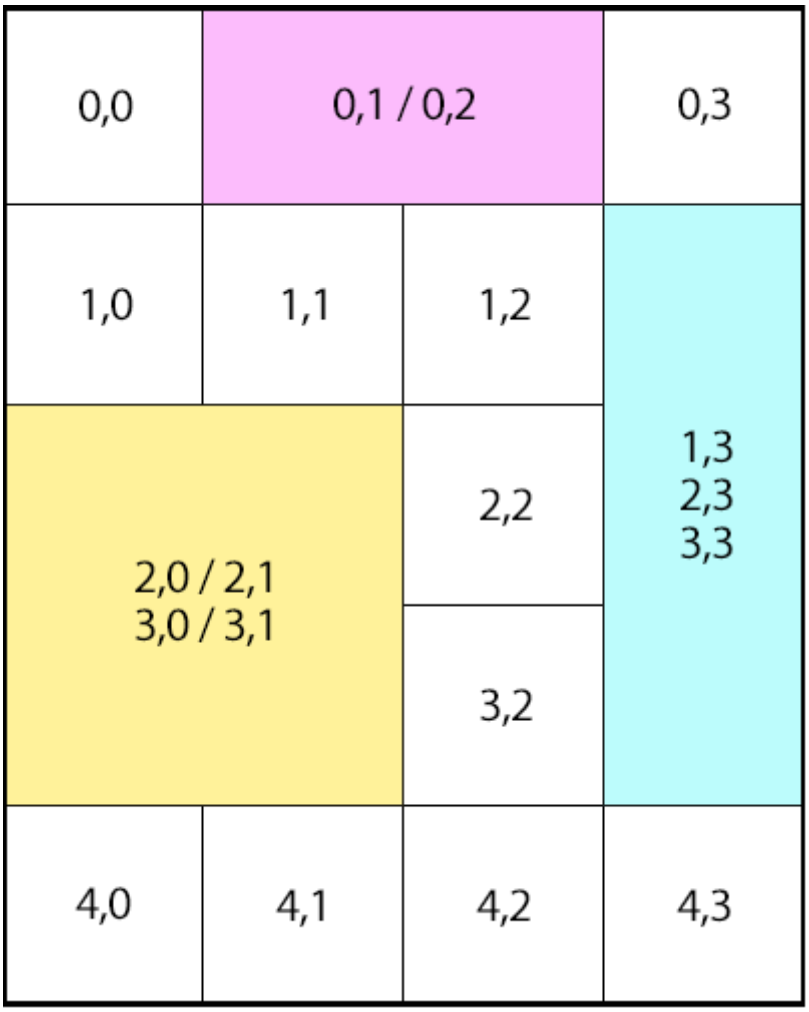
The bitmaps in the left column have different **alignments** (Top-Left, Centered, Bottom-Right). In the inner grid 9 bitmaps are scaled down (by the stretching property) to a fixed element size.

> The grey grid raster in the background is set up manually by solid nodes to illustrate the inner grid borders.

Cell Spanning Example

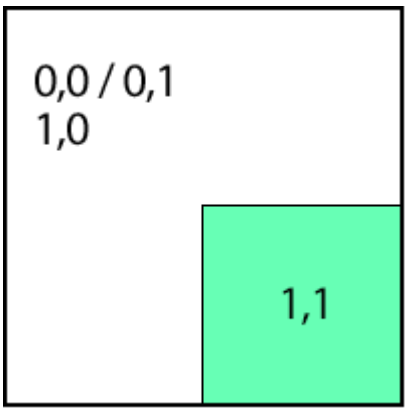
In the example, below a Candera::GridLayouter with 5 rows and 4 columns defined is used to exemplify how cell spanning is working:

- **Horizontal Spanning**: Element in Row(0) and Column(1) has a Column Span of 2.
- **Vertical Spanning**: Element in Row(1) and Column(3) has a Row Span of 3.
- **Horizontal and Vertical Spanning**: Element in Row(2) and Column(0) has a Row Span of 2 and Column Span of 2.



> Please note that single cells might still be used for other elements if needed, like in the following example:

> A Candera::GridLayouter with 2 rows and 2 columns is defined. Element in (0,0) has a row and column span of 2, but element at (1,1) is still accessible.



See also:

- Grid Layouter in SceneComposer User Manual

Candera::BaseLineLayouter

Introduction

Candera::BaseLineLayouter behaves similar to a Horizontal StackLayouter but instead of aligning the objects to the client area of the Layouter, it aligns them to a line.

The idea is to be able to align text with different sizes in different ways.

- For instance it can align all the text by the top of its bounding box (VTop).
- It can align the text by taking into consideration the bottom of the bounding box (VBottom).
- Or, the most interesting feature it can align the items with different size by their **base line** (VCenter).

Example

The **Layout2DSolution** provided in the content folder of *cgi_studio_player* gives an example how the 2D BaseLineLayouter can be applied in SceneComposer.

- In the Scene2DBaseLineLayouter scene, on three groups, a Candera::BaseLineLayouter is defined.
- The base line offset configured on the groups is visualized by an orange line.
- The text parts of each group are defined with "VTop", "VBottom", or "VCenter", for the Vertical Alignment attribute.

The baseline can be either fixed or automatic. The automatic baseline is configured such that all the children fit from the top of the client area.

See also:

- Baseline Layouter in SceneComposer User Manual

Candera::OverlayLayouter

Introduction

A Candera::OverlayLayouter is a container that layouts enfolded nodes by overlaying them in same place in a sorted sequence. With this layouter, elements can be arranged that they fulfill a given area and lay on top of one another.

Example

The **Layout2DSolution** provided in the content folder of *cgi_studio_player* gives an example how the OverlayLayouter can be applied in SceneComposer. Some important use cases are:

- The outer group represents the given area, in which the nodes are arranged. You can set the size of the group to define the height and the width of the area. If you don't set any size, it will automatically be set to the size of the biggest inner child element.
- The sequence of the nodes in the overlay group is important. In the example the Inner_Overlay_Group is displayed in front of the Outer_Overlay_Group.
- Three very small images (mainBmp, gradient_verticalBmp, gradient_horizontalBmp, see in ExampleSolution) are stretched and blended to create a big image. They fulfill the given area.
- The text nodes are aligned to the center and in the corners. You can set several vertical and horizontal alignments for each node.

See also:

- Overlay Layouter in SceneComposer User Manual

Candera::DockPanelLayouter

Introduction

The Candera::DockPanelLayouter provides an easy docking of elements to the left, right, top, bottom or center of the panel. To dock an element to the center of the panel, it must be the last child of the panel.

Example

The **Layout2DSolution** provided in the content folder of *cgi_studio_player* gives an example how the DockPanelLayouter can be applied in SceneComposer.

See also:

- Candera::DockPanelLayouter
- DockPanel Layouter in SceneComposer User Manual

Automatic Resize Depending on Content

Introduction

This tutorial will show how to create resizable dialog popups using a hierarchy of groups and different kind of nodes (bitmap, solid color and text) arranged specifically using layouters.

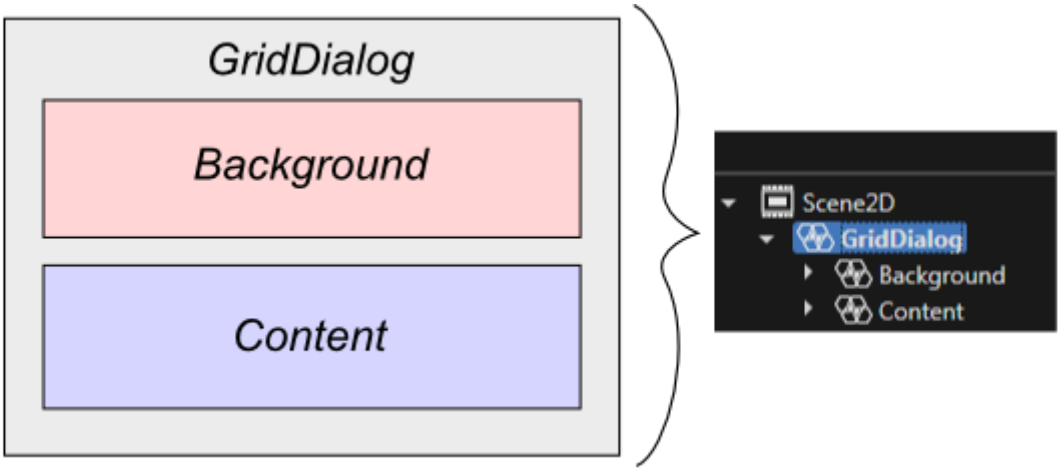
Examples

Resizable Dialog Example Using Grid Layouter

For this example the dialog elements are arranged using Grid and Overlay layouts.

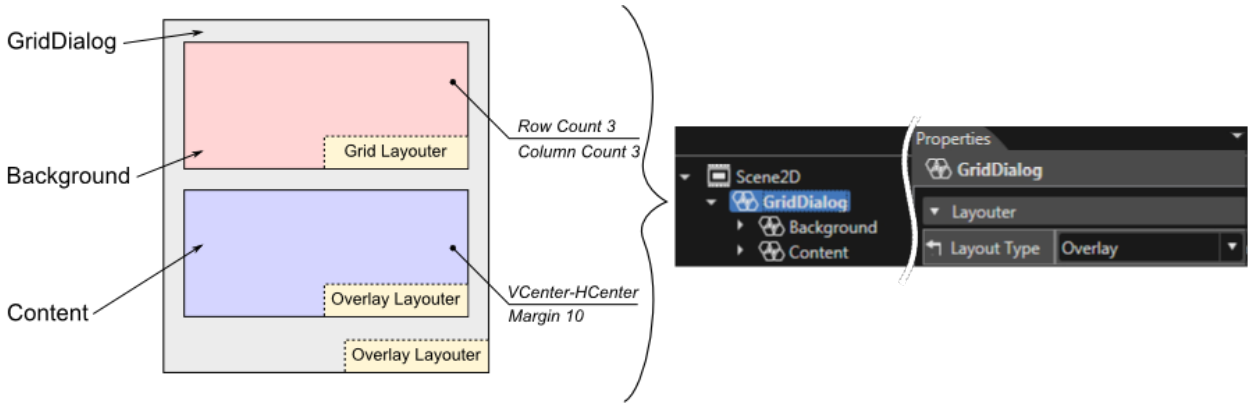
Group Hierarchy

Create the parent group for the pop-up and name it GridDialog. Create another two groups (Background and Content) and add them as children of GridDialog group.

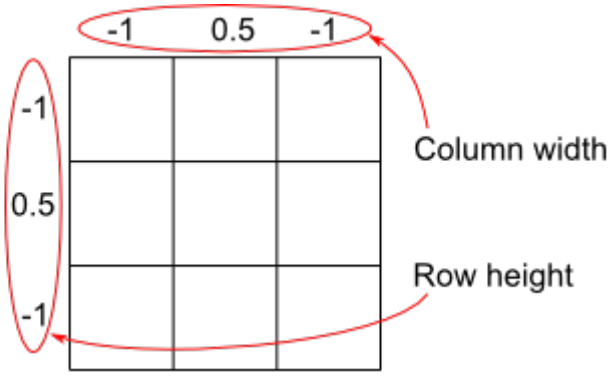


Layout Configuration

Set the layout for each group as shown in the image below:



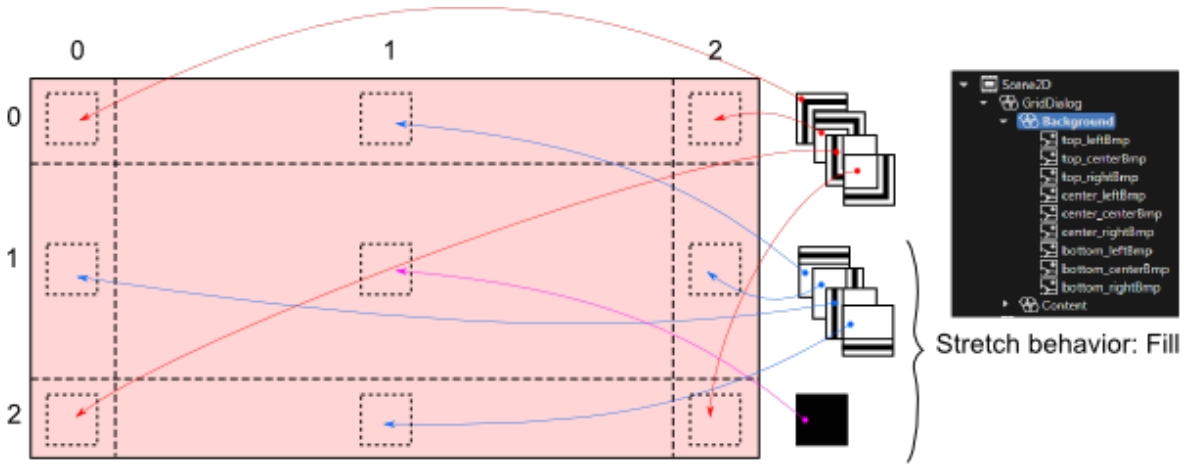
Configure the column width and row height:



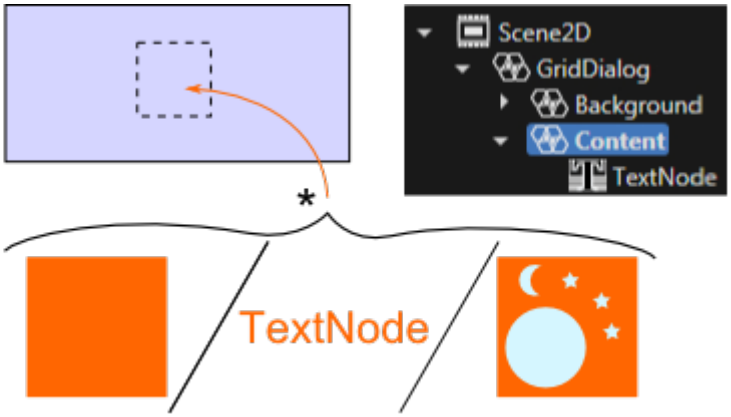
In SceneComposer this can be achieved in the "Grid Layout Editor" (right click on the Background group and select "Configure Layout").

Adding Nodes

The Background group should be populated with bitmap or solid color nodes with the following settings:



The Content group might be filled with one or more nodes of any type, as desired:



Result

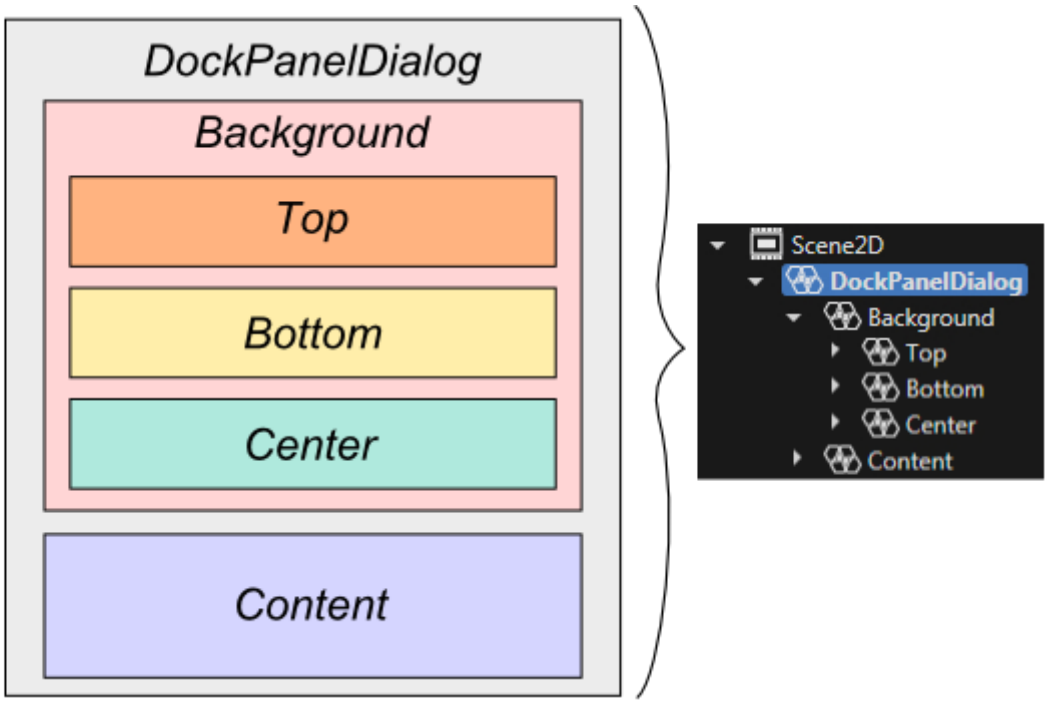
After putting everything together the dialog will look as above:

Resizable Dialog Example Using DockPanel Layouter

For this example the dialog elements are arranged using DockPanel and Overlay layouts.

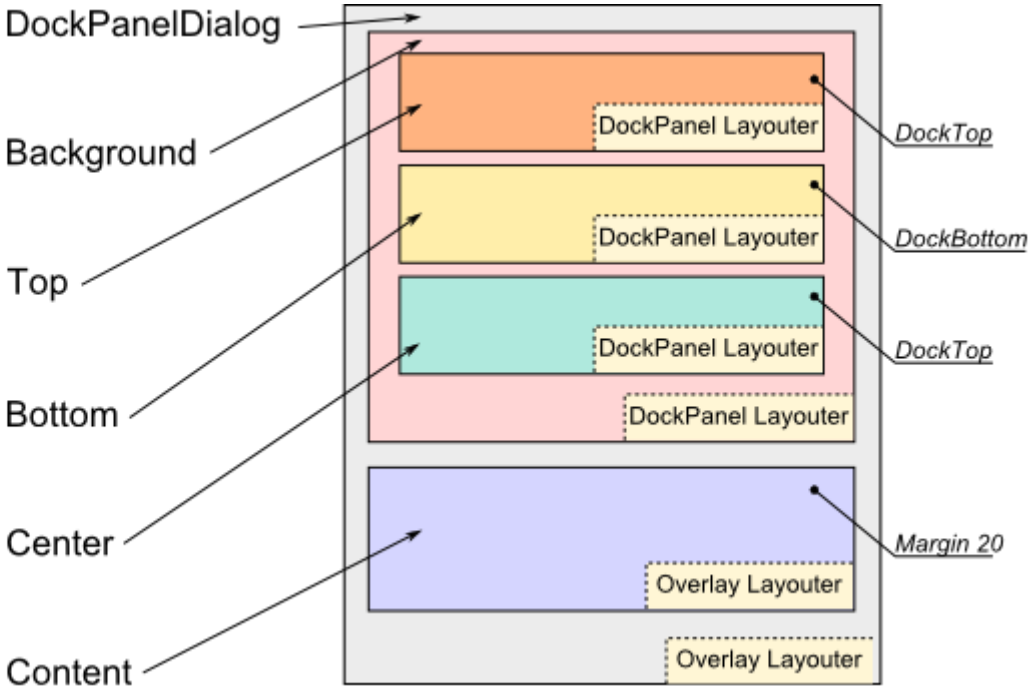
Group Hierarchy

Create the following structure of groups:



Layout Configuration

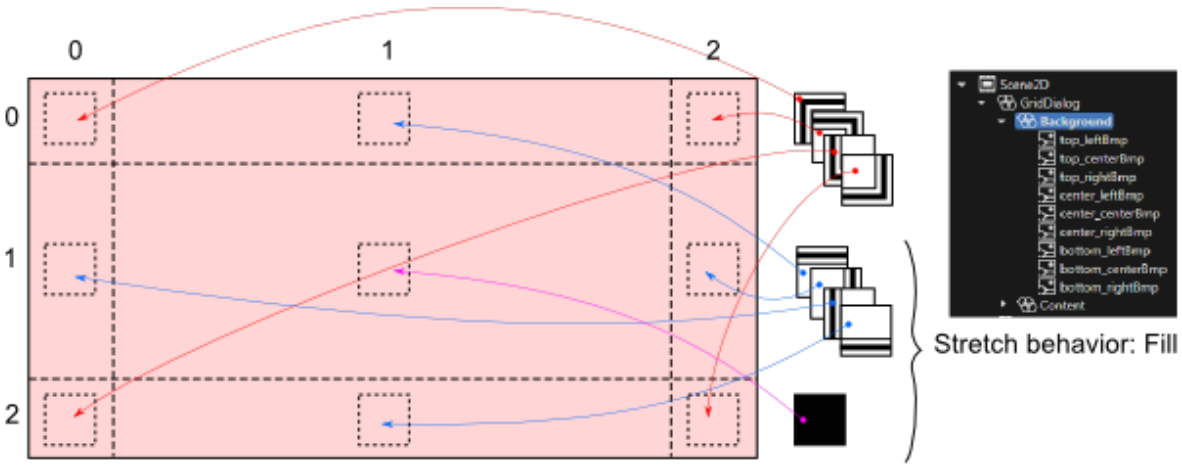
Set the layout for each group as shown in the image below:



> Be sure the order of adding the children groups in the Background group is Top - Bottom - Center otherwise the elements will not be arranged correctly.

Adding Nodes

The children groups of Background should be populated with bitmap or solid color nodes with the following settings:



> Be sure the order of nodes in the groups is Left - Right - Center otherwise the elements will not be arranged correctly.

The Content group should be configured as explained in the previous chapter for the dialog based on grid layout with the only difference that the text node will be replaced with a bitmap. Please see the Content group part from the Adding Nodes section.

Result

After putting everything together the dialog will look as above:

Candera 2D Listener

Applications can receive notifications on scene graph events. Therefore implement a listener, which defines hooks to receive those notifications.

Following listener interfaces are provided in Candera 2D:

- Candera::Camera2DListener defines hooks that are called before or after a Camera2D is rendered. In order to register a Camera2DListener to a Camera2D simply derive from Camera2DListener and override pure virtual functions with custom code.
- Candera::Renderer2DListener defines hooks that are called before or after a node is rendered. It supports events for OnNodePreRender and OnNodePostRender, so you are informed before and after a node is rendered.

Example Listener Implementation

Refer to Using Animation Callback Functions for an example how to implement and use an Animation listener.